

Sheet1

#	Pri.	Story	Example	Test
1		As a programmer, I want to print out all of the strings that are passed as parameters to my program so I can see what they are.	If the program is called as <code>./stack this is a test</code> , it will print: <code>argv[1] = this</code> <code>argv[2] = is</code> <code>argv[3] = a</code> <code>argv[4] = test</code>	<pre>Stacks-story1> ./stack this is a test argv[1] = this argv[2] = is argv[3] = a argv[4] = test execute() called</pre>
2		As a user, I want my program to notify me when I enter an incorrect parameter and tell me what the correct parameters are, and to notify me if I give a <code>-h</code> parameter, so I do not need to remember all of the parameters.	The call <code>stack -h</code> causes the help message to be printed and the program to execute; <code>stack -s</code> causes the help message to be printed and the program to terminate; <code>stack</code> causes the program to execute. <code>xs</code>	<pre>Stacks-story2> ./stack -h stack: parameters are -h: print this message execute() called Stacks-story2> ./stack -x Error: unknown parameter x stack: parameters are -h: print this message Stacks-story2> ./stack execute() called</pre>
3		As a programmer, I want to be able to push a character on a stack, so I can use the stack to match parentheses.	After function <code>"push(stack *s, char x)"</code> is the stack will have the value <code>x</code> as its top position.	<pre>Test 3: push pushed a top: a :bottom pushed b top: b a :bottom pushed c top: c b a :bottom</pre>
4		As a programmer, I want to pop a character from a stack so I can use the stack to match parentheses.	The function <code>"push(stack s, char x)"</code> will push a character onto the top of the stack; the function <code>"char pop(stack s)"</code> will pop an integer off the top of the stack.	<pre>Test 4: pop pushed a top: a :bottom pushed b top: b a :bottom pushed c top: c b a :bottom popped c after pop top: b a :bottom popped b after pop top: a :bottom popped a after pop top: :bottom Error: pop(s, *val) empty stack s popped ? after pop top: :bottom</pre>

Sheet1

5		As a user, I want to be able to enter a set of brackets to see if they are matched at the command line, so I can use the program to match brackets.	If you enter "stack -b {}". the program will print out that the bracket match. If you type "stack -b "{}()" it will point to the mismatched bracket.	<pre> Test 5: match_brackets () Brackets match ([^ mismatch [] Brackets match {} Brackets match (a + b) Brackets match {[x * (a + b)] } Brackets match {[x * (a + b)] } ^ mismatch </pre>
6		As a programmer, I want to reimplement a stack that contains integers, so I can write the function to evaluate postfix notation.	Calling "push(stack s, int i)" will push integer i onto the stack; calling "pop(stack s)" immediately afterwards will pop the integer and return the value	<pre> Test 6: integer stack pushed 1 top: 1 :bottom pushed 2 top: 2 1 :bottom pushed 3 top: 3 2 1 :bottom popped 3 after pop top: 2 1 :bottom popped 2 after pop top: 1 :bottom popped 1 after pop top: :bottom Error: pop(s, *val) empty stack s popped -1 after pop top: :bottom </pre>
7		As a programmer, I want to implement a program that will evaluate expressions based on postfix notation, so I can see how it is done.	If you enter "stack -p 32+" the program will print out 5.	<pre> Test 7: evaluate_postfix eval_postfix("1 2 +"): 3 eval_postfix("1 2 -"): -1 eval_postfix("3 2 *"): 6 eval_postfix("4 2 /"): 2 eval_postfix("20 4 +"): 24 eval_postfix("2 40 +"): 42 eval_postfix("20 40 +"): 60 eval_postfix("10 20 40 + -"): -50 eval_postfix("10 20 40 + /"): 0 eval_postfix("10 20 + 40 -"): -10 eval_postfix("10 20 * 40 -"): 160 </pre>